# CS 340 Project Two README

**Christopher Ronca**

## About the Project

The Grazioso Salvare Dashboard is a web application designed to provide an interactive interface for analyzing animal shelter data. The dashboard allows users to filter and visualize data related to animal rescues, helping Grazioso Salvare make informed decisions about animal rescue operations.

## Motivation

The project was developed to assist Grazioso Salvare in efficiently managing and analyzing large datasets of animal shelter outcomes. By providing a user-friendly dashboard, the organization can quickly access and interpret data, leading to better resource allocation and rescue planning.

## Getting Started

To get a local copy up and running, follow these simple steps:

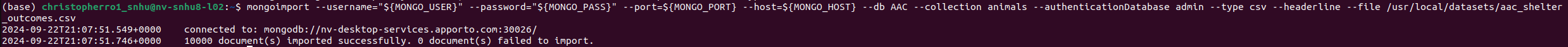
1. Ensure all required software is installed on your system (see below for comprehensive listing).
2. Configure your MongoDB instance to support authentication.
3. Import the dataset aac\_shelter\_outcomes.csv into a collection within a database on the MongoDB server.
4. Update the connection details in the code with your MongoDB credentials.

**Required Software:**

* **MongoDB v6.x**: Provides NoSQL database functionality for handling large datasets. Chosen for its performance and scalability.
* **Jupyter Notebooks v6.5.2**: Offers a front-end interface for developing and testing the dashboard.
* **Python v3.9**: Selected for its robust libraries and ease of integration with MongoDB.
* **PyMongo v4.3.3**: A Python module for interfacing with MongoDB, enabling CRUD operations.
* **CRUDMongoModule v1**: Custom module for performing CRUD operations using PyMongo.
* **Dash Framework v2.8.1**: Used for creating the interactive dashboard and managing the web application.
* **Pandas v1.5.3**: A data analysis library for manipulating datasets and preparing data for visualization.

## Installation

1. ***MongoDB Setup****:*
   * *Install MongoDB and configure it to support authentication.*
   * *Import the dataset aac\_shelter\_outcomes.csv into a collection named animals within a database named AAC. (see below for a command-line import example)*



1. ***Python Environment****:*
   * *Install Python 3.9 and set up a virtual environment.*
   * *Install the required Python packages using pip:*

*pip install pymongo pandas dash jupyter-dash*

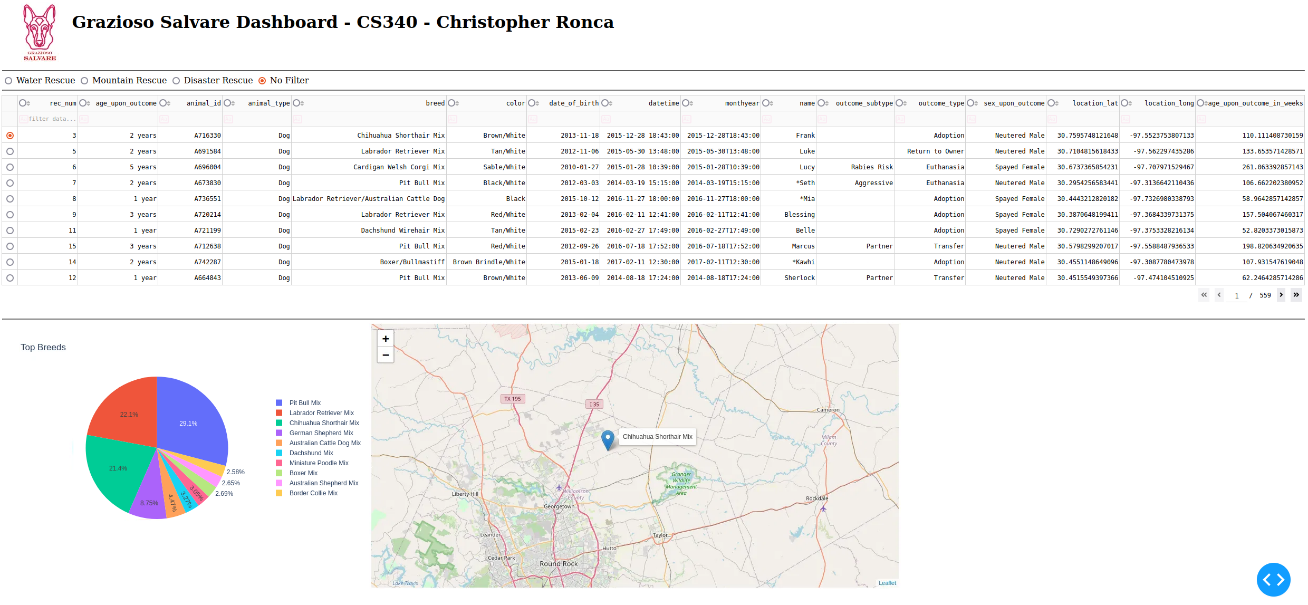
1. ***Configure the Application****:*
   * *Update the connection details in the AnimalShelter class with your MongoDB credentials (****NOTE:*** *this will require your username, password, host, port, database, and collection name).*

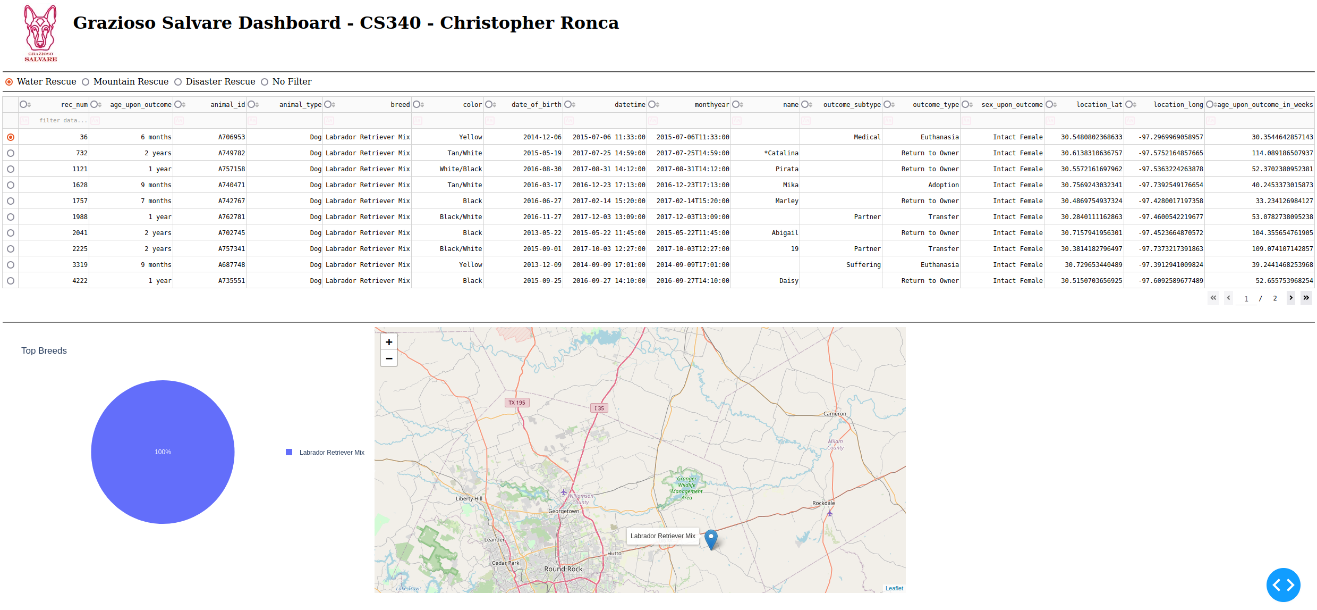
## Usage

The dashboard provides several interactive features, including:

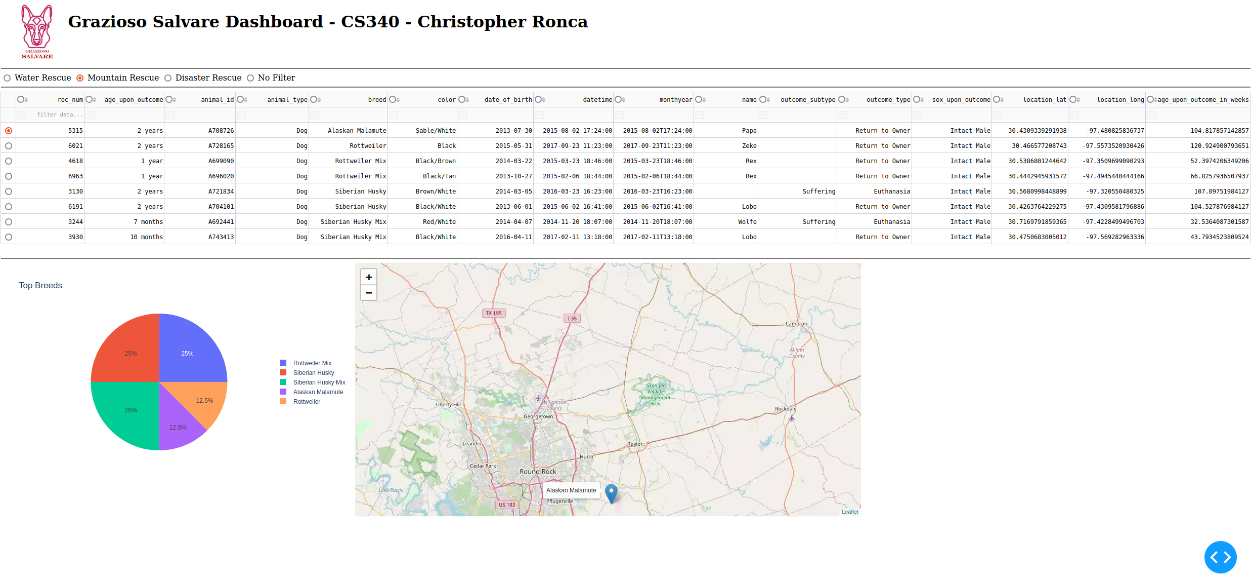
* Filtering data by rescue type (Water, Mountain, Disaster, No Filter).
* Viewing data in a table format with sorting and filtering capabilities.
* Visualizing data through pie charts to show the distribution of animal breeds.
* Displaying geolocation data on a map for selected entries.

The landing page will default to the “No-Filter” option and show all returns from the read function:

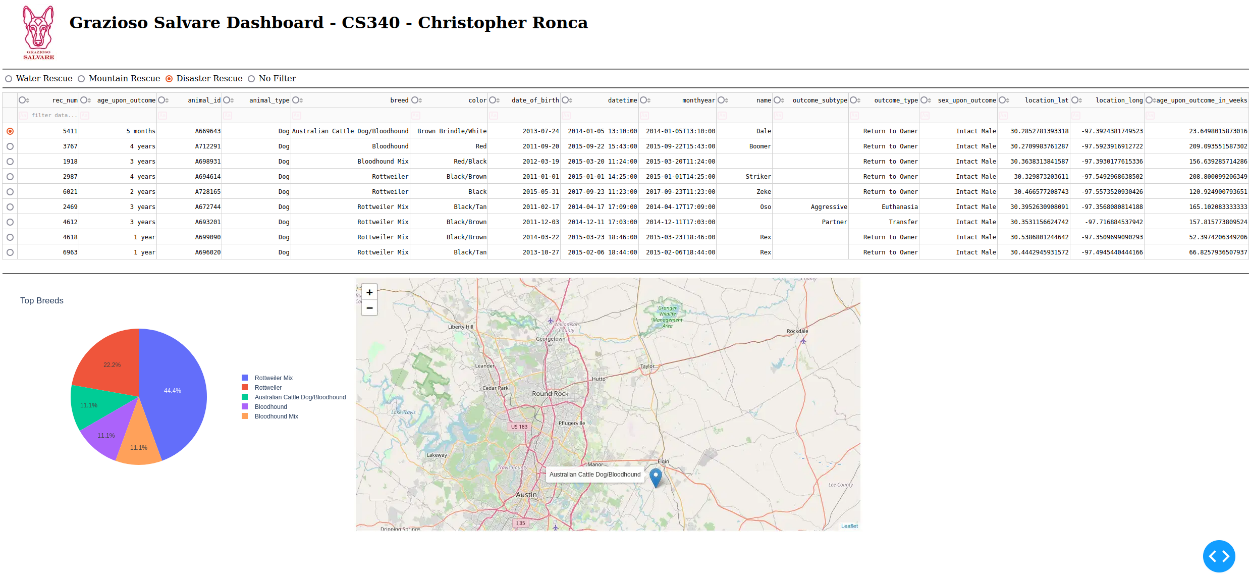


Likewise, each filter option will display data accordingly:  
  
Water Rescue:  


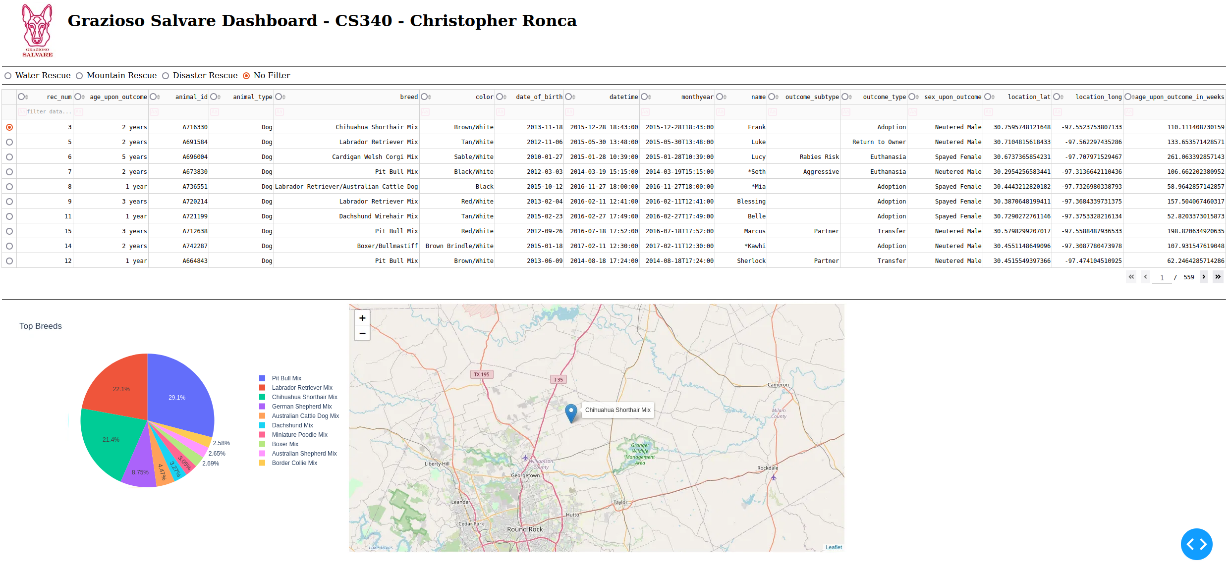
Mountain Rescue:



Disaster Rescue:



No Filter:



The CRUD module (aac\_CRUD.py) provides four main functionalities: creating, reading, updating and deleting document(s) in a MongoDB collection.

A screenshot of a computer code

Description automatically generated

The ‘*create’* method is responsible for inserting a document into a MongoDB collection. It takes a dictionary data as a parameter, which contains the key-value pairs representing the document to be inserted. The method first checks if data is not None, raising an exception if it is empty. It then attempts to insert the document using insert\_one(data), which is part of the pymongo library. If the insertion is successful, it returns result.acknowledged, a boolean indicating the operation was acknowledged by the server. If an exception occurs during the insertion, it catches the exception, prints an error message, and returns False.

The ‘*read’* method is designed to query documents from a MongoDB collection based on a specified query, which is a dictionary containing the criteria for document retrieval. It uses a try-except block to handle potential errors. Within the try block, it calls self.collection.find(query) to search for documents matching the query criteria, returning a cursor. This cursor is then converted to a list and returned, containing all matching documents. If an exception occurs during the query operation, the method catches the exception, prints an error message, and returns an empty list, ensuring robust error handling.

The ‘*update’* method is responsible for modifying existing documents in a MongoDB collection based on a specified query. It takes two parameters: query, a dictionary defining the criteria for selecting documents to update, and update\_data, a dictionary containing the fields to be updated and their new values. The method uses update\_many to apply the updates to all documents matching the query. It returns the number of documents modified. If an exception occurs during the update operation, it catches the exception, prints an error message, and returns 0, ensuring robust error handling.

The ‘*delete’* method is designed to remove documents from a MongoDB collection based on a specified query. It takes a single parameter, query, which is a dictionary defining the criteria for selecting documents to delete. The method uses delete\_many to remove all documents matching the query. It returns the number of documents deleted. If an exception occurs during the deletion operation, it catches the exception, prints an error message, and returns 0, ensuring robust error handling.

### Tests

To test the dashboard, run the Jupyter Notebook and execute the cells to start the Dash server. Verify the functionality by interacting with the dashboard and checking the data visualizations.

## Roadmap/Features (Optional)

* Implement additional filters for more specific data analysis.
* Enhance the visualization options with more chart types.
* Integrate real-time data updates from the MongoDB database.

## Development Hurdles

## Challenges

## Data Integration: Integrating MongoDB with the Python application required careful handling of data types and structures, especially with MongoDB's BSON format.

## Interactive Visualization: Creating a responsive and interactive dashboard that could handle large datasets efficiently was challenging.

## Solutions

## Using PyMongo: Leveraging PyMongo's capabilities to handle BSON data types and perform efficient queries helped overcome data integration challenges.

## Dash Framework: Utilizing Dash's powerful components for building interactive web applications allowed for the creation of a responsive and user-friendly dashboard.

## Contact

Christopher Ronca